Managing key multicasting through orthogonal systems

José Antonio Alvarez-Bermejo and Juan Antonio Lopez-Ramos,
University of Almeria, Spain
Joachim Rosenthal and Davide Schipani,
University of Zurich, Switzerland

Abstract—**In this paper we propose a new protocol to manage multicast key distribution. The protocol is based on the use of orthogonal systems in vector spaces. The main advantage in comparison to existing multicast key management protocols is that the length and the number of the messages which have to be sent is considerably smaller. This makes the protocol attractive when the number of legitimate receivers is large.**

Keywords: Multicast key management, data transmission security

I. INTRODUCTION

Traditional security measures are mainly applicable to a unicast environment, i.e. communications take place between two single parties. For instance, data confidentiality, one of the most important features in network security, can be offered in this environment by means of a pair of keys. However there exist many different situations where the usual secure unicast protocols cannot be used, mainly due to the nature of the information to be transmitted. This usually happens when trying to deliver data from a sender to multiple receivers, especially when a huge amount of data needs to be delivered very quickly. One of the most efficient ways to do this is the so-called multicast. In a multicast protocol a certain group of people receives the information and this group is usually highly dynamic. In a typical situation users join and leave the group constantly ([10]).

There are a number of exciting multimedia applications that make good use of multicast capability, such as stock quote services, video-conferencing, pay-per-view TV, Internet radio, and so on. Many of these multicast applications require security in data transmission, i.e., data can only be exchanged among an exclusive group of users. In multicast communications a situation of "many-to-many" can also take place if several clients, or all, act as a source of data. Multiconferences are an example of this: in that situation each data source establishes a one-to-many multicast communication.

The typical approach to establish secure multicast communications is to agree on one or several symmetric encryption keys in order to encrypt messages. However, the key, or keys, must be renewed periodically to prevent outer or inner attacks.

Depending on how key distribution and management are carried out, secure multicast schemes are divided into centralized and distributed schemes. Centralized schemes depend directly on a single entity to distribute every cryptographic key. Distributed schemes are able to manage higher number of audiences but, on the other hand, key management involves other problems that make them more complex ([10]). In the following lines we recall some centralized schemes for key management.

A very well-known protocol is *Hierarchical Tree Approach* (HTA) [15]. It uses a logical tree arrangement of the users in order to facilitate key distribution. The benefit of this idea is that the storage requirement for each client and the number of transmissions required for key renewal are both logarithmic in the number of members. Other key tree approaches and extensions are LKH [17], LKH++ [3], OFT [13] or ELK [9].

In [2] the so-called *Secure Lock* protocol is introduced. The authors approach the problem in a computational manner and make use of the Chinese Remainder Theorem instead of a tree arrangement. Its main drawback is the large computational cost required at the key server side on each rekeying operation: the computing time needed becomes quickly problematic as the number of members grows [7].

In [11], a divide-and-conquer extension of the Secure Lock is proposed. It combines the Hierarchical Tree Approach and the Secure Lock: members are arranged in a HTA fashion, but the Secure Lock is used to refresh keys on each tree level. Therefore, the number of computations required by the Secure Lock is reduced.

Another computational approach is introduced in [6] with the particular application on Pay-TV but extendable to any other secure multicast application. The idea is to use polynomials over a finite field interpolating hashes

of secret values belonging to the authorized users. The main drawbacks are the large size of the polynomials involved and that the hash function must be renewed with any rekeying operation, due to security concerns.

More recently in [8] the authors introduce another solution based on the Extended Euclidean Algorithm. Throughout this paper we will refer to this protocol as *Euclides*. The server distributes a secret via the inverse of an integer modulo a product of coprime secret numbers, each one of them belonging to an authorized user. The authors show that an old user could try a factorization attack, which forces to consider prime numbers of an adequate size. The length of the messages grows linearly with the number of users, so that if this number is huge, users might be forced to be distributed in groups.

The distribution by groups is in fact often beneficial and is used by most key managing protocols. A first benefit is the parallelization of the process which speeds up the rekeying operations. Secondly a compromised key in one of the groups does not affect the others. Last but not least, in most applications of secure multicast the group distribution is connected with the scalability of the system, i.e., the efficiency of the communication protocols concerning the rekeying process, with particular reference to leave and join operations. Groups are usually highly dynamic and the joining or the leaving of users implies a rekeying operation, and thus key refreshment due to this fact in one group does not affect the others.

In the next Section we describe a new protocol, analyse its security, and compare it with other existing protocols. In Section 3 we show that it can also be used for authentication between users. Sections 4 and 5 show an implementation of the protocol to prove that it is scalable and efficient.

II. THE PROPOSED PROTOCOL

Let the potential users be denoted with the integers $1, \ldots, n$

1) Initialization step:
   Let $\mathbb{K}$ be a field and V be a $\mathbb{K}$ vector space of dimension $m \geq n$ (see also next subsection for the choice of m). Let $<,>$ be a bilinear form which we assume to be nondegenerate and symmetric. Let $B = \{e_1, \ldots, e_n\}$ be a set of n mutually orthogonal vectors in V having the property that $< e_i, e_i > \neq 0$ for $i = 1, \ldots, n$. We select a family $\{x_i\}_{i=1}^n$ of random nonzero scalars in $\mathbb{K}$. Note that $B' = \{x_1 e_1, \ldots, x_n e_n\}$ spans the same subspace as B. These two sets are kept secret by the server and each user i is assigned the vector $v_i = x_i e_i$. By our assumptions we know that $< v_i, v_i > \neq 0$ for $i = 1, \ldots, n$.

2) Sending the information:
   Suppose that we want to distribute the secret $s \in \mathbb{K}$. Then we compute the vector $x_1 e_1 + \cdots + x_n e_n$ and multicast (broadcast) $c = s(x_1 e_1 + \cdots + x_n e_n)$.

3) Recovering the information:
   Each user computes $h = << c, v_i >> = s < v_i, v_i >$. The secret s is then recovered by computing $s = h < v_i, v_i >^{-1}$.

4) Key refreshment:
   a) Join:
      If user j joins, then she is assigned one of the vectors in B' that is not being used by another user, say $x_j e_j$. The server selects a new secret $s' \in \mathbb{K}$ and multicasts $c' = s'(x_1 e_1 + \cdots + x_n e_n)$.
   b) Leave:
      If user j leaves, then her vector $v_j = x_j e_j$ is deleted from B' and a new set B'' is considered formed by the same vectors in B' but substituting v_j with $v_j' = x_j' e_j$ where $x_j' (\neq x_j)$ is selected at random in $\mathbb{K}$. A secret s' is then distributed as before.

A. Security

We first notice that, by choosing m appropriately, we can be sure that there are sufficiently many n-tuples of mutually orthogonal vectors in V, so that a brute force attack to find B is not feasible. If $\mathbb{K}$ is a finite field $\mathbb{F}_q$, we can for example choose $m > 2n$ and we know that there are more than

$$(1 + o(1))\frac{q^{\frac{3}{2}n^2}}{n!}$$

n-tuples of mutually orthogonal vectors in V ([5], [14]). We remark also that any of these vectors should not be orthogonal to itself, otherwise there would be a problem to retrieve the secret. This is readily seen not to be a weakness with respect to brute force attacks, as soon as the characteristic of the field is big enough, or 0.

Assume the set B is known, instead of being kept secret. Since B is a linearly independent set, one can compute readily the unique coefficients $z_1, \ldots, z_n$ such that

$$c = z_1 e_1 + \cdots + z_n e_n.$$

An authorized user knowing the vector $v_j = x_j e_j$ and having computed $z_j e_j$ readily computes x_j and s from $z_j = s x_j$. With this all the private numbers x_i, $i = 1, \ldots, n$ can be readily computed by this user. Such a user would have the chance to use this later in her own interest. As it is often the case, inner attacks are more dangerous than outer ones.

The security is clearly compromised not only if the set B is made public, but also if just one vector of B' becomes known to unauthorized users: in fact getting s

involves knowing at least one vector in the set B' used to compute c. We can think at different ways for an attacker to get such a vector.

First an old user can try to get the new s' using her old vector, say v_i. If she multiplies $< c', v_i >$, then she would get

$$< c', v_i > = < s'(x_1 e_1 + \cdots + x'_i e_i + \cdots + x_n e_n), x_i e_i >$$
$$= s' x_i x'_i < e_i, e_i > .$$

But now she would have to know either the vector e_i or (equivalently) the value x_i to get the new secret s'.

Another option consists in trying to derive some information from the difference between two different rekeying messages c and c'. But

$$c - c' = (s - s')x_1 e_1 + \cdots + (s x_i - s' x'_i)e_i + \cdots + (s - s')x_n e_n$$

Then $< c - c', v_i > = (s x_i x_i - s' x'_i x_i) < e_i, e_i >$ and as before, the secrecy of e_i avoids leaking any information on s'. And even if the new user shares s' with the attacker, they do not get any information concerning x_i, x'_i and so e_i is not in the risk of being compromised.

A similar situation occurs when trying to make a plain-text chosen attack. This corresponds to an authorized user trying to derive some information from two different rekeying messages. In this case, $< c - c', v_i > = (s - s')x_i^2 < e_i, e_i >$. Again no information on e_i can be obtained.

Finally we can foresee an attack based on the collection of many subsequent pieces of information. Namely, anybody observing the information flow could get n linearly independent key refreshments $c_1, \ldots, c_n$. Note that this is the case whenever a user i leaves and in that case, the set $B' = \{x_1 e_1, \ldots, x_i e_i, \ldots, x_n e_n\}$ would change to $B'' = \{x_1 e_1, \ldots, x'_i e_i, \ldots, x_n e_n\}$. Now, suppose without loss of generality that $n = m$; if the server sends $(s_i x_1, \ldots, s_i x_n)$ as a rekeying message $c_i = (c_{i,1}, \ldots, c_{i,n})$, then we would consider the matrix

$$M = \begin{pmatrix} c_{1,1} & \cdots & c_{n,1} \\ \vdots & \vdots & \vdots \\ c_{1,n} & \cdots & c_{n,n} \end{pmatrix}$$

where each column $(c_{i,1}, \ldots, c_{i,n})$ represents the coordinates of the refreshment c_i with respect to B (as $c_{i,j} = s_i x_j$ for $i, j = 1, \ldots, n$); then M represents the change of basis from the basis $C = \{c_1, \ldots, c_n\}$ to B. The inverse of M will reveal then B in terms of the basis C. And knowing a pair (s, c) would compromise all the secrets (of the other users) used to get this pair (s, c), as noted at the beginning of this subsection.

Therefore it is convenient that B is chosen not to be the canonical basis used to represent vectors of the vector space V, so that what is sent by the server is not plainly $(s_i x_1, \ldots, s_i x_n)$.

Let us illustrate this with the following easy example:

Example:

Let $B = \{(1, 1, 1), (1, -2, 1), (-1, 0, 1)\}$ be an orthogonal basis of the euclidean vector space $\mathbb{R}^3$ (with the usual scalar product $<, >$) and assume $x_1 = 2$, $x_2 = 3$, $x_3 = 5$. Then $B' = \{(2, 2, 2), (3, -6, 3), (-5, 0, 5)\}$.

If we want to rekey with $s = 4$, then we have to multicast $c_1 = 4(2, 2, 2) + 4(3, -6, 3) + 4(-5, 0, 5) = (0, -16, 40)$.

User 1 can recover s by calculating

$$h = < (0, -16, 40), (2, 2, 2) > = 48,$$

and then

$$s = h < (2, 2, 2), (2, 2, 2) >^{-1} = 4.$$

Users 2 and 3 act similarly.

Now suppose that user 2 leaves and x_2 is changed to $x_2 = 2$. Then the rekeying message for $s = 3$ is $c_2 = 3(2, 2, 2) + 3(2, -4, 2) + 3(-5, 0, 5) = (-3, -6, 27)$. Finally, suppose that user 1 leaves, x_1 becomes 3 and the new secret s is 2, so that $c_3 = 2(3, 3, 3) + 2(2, -4, 2) + 2(-5, 0, 5) = (0, -2, 20)$. Now the basis given by $\{c_1, c_2, c_3\}$ does not tell anything about the basis B.

Remark: It should be remarked that, if we restrict ourselves to work in a subring of the base field that admits an algorithm to compute GCDs, then s divides the GCD of the coordinates of c. Observe, for instance, that in the Example $s = GCD(0, -16, 40)$ and after the first rekeying $s = GCD(-3, -6, 27)$. Thus this situation should be avoided for a security issue.

B. Comparison with other schemes

We compare here our new proposal with some of the other key managing protocols existing in the literature and cited in the introduction. The main parameters we will focus on are the key storage cost and the length of the messages. For additional comparisons, we refer to [8], in particular Table 1, where Euclides is compared with previous protocols and other features are also taken into account.

As for the protocol we are introducing in this paper, the server has to store one scalar per user, the x_i's, and an orthogonal system, B, for each considered group, while the length of the rekeying messages is $n \cdot C$, where C is either the bit-length of the elements in a finite field $\mathbb{K}$ or an upper bound for this bit-length that can be selected during the initialization step in case we are considering a field whose characteristic is 0.

In [8], Euclides was introduced and shown to be already very competitive with respect to existing protocols, however the present proposal offers some additional advantage concerning the length of the messages.

In Euclides, in fact, the key storage cost can be of the same order as here, but the length of the messages could become a problem unless some key management by groups is used. In fact, by security requirements every private key held by any user, an integer, has to be of appropriate length to avoid a factorization attack by an old user (cf. [8]). In this way integers of length 1024 bits onwards should be considered and since the rekeying messages are of the same order as the product of all these integers, then for large groups these could be unaffordable. On the other hand, in this new protocol messages can be considerably shorter than in Euclides, depending on the number of users in the group and on the cardinality of the field chosen for the scalars (or eventually, as noted before, on bounds we could fix for the scalars in use in case of fields of characteristic 0).

Suppose for example that we are dealing with a field of the order 64-bits length elements and we are using a vector space of dimension 10000. Then rekeying messages would be shorter than 80Kbytes length, which is perfectly affordable by any multicast network used for this purpose. In the case of Euclides, using primes of 64-bits length produces messages of the same length, i.e. 80Kbytes. However, any user, as it is shown in [8], has access to a multiple of the product of all the secret keys and so this bit-length of the primes is not enough for a secure rekeying process since a factorization attack would succeed very quickly. To avoid this we are forced to deal with 1024-bits length primes (at least). This leads to over 1Mbyte length rekeying messages. Otherwise we have to divide this audience in at least 12 groups in order to deal with messages of length comparable to that of the new proposal.

In the case of Secure Lock each user holds a pair of keys, an integer, R_i and a symmetric key, k_i. The server encrypts the secret using the symmetric key k_i of every user, obtaining a number for each one of them, N_i. Then the server solves the congruence system $x \equiv_{R_i} N_i$ and multicasts the solution U of this system. We observe that, as in the case of Euclides, the length of the messages is of the same order as the product of all the integers R_i and that with every refreshment a congruence system has to be solved, which can quite slow down the rekeying process. Recall also that the server has to encrypt as many times as the number of authorized users. In order to speed it up it is commonly used jointly with HTA. However the length of rekeying messages still depends on the users involved in each group.

As far as the Conditional Access Service introduced in [6] is concerned, amid a good behavior regarding key storage, the high degree of the polynomials involved again generally forces a partition of the users into groups. Moreover the hash function used to create the interpolator polynomial that is used to distribute the secret has to be changed with every rekeying process, as mentioned above.

In our case, the rekeying process only requires four simple operations, namely a scalar substraction (say $x'_i - x_i$), a multiplication of a scalar by a vector, $(x'_i - x_i)v_i$, a vector addition and finally, the computation of a multiple of the output according to the secret to be distributed, which is considerably faster with respect to all the previously considered protocols.

III. AUTHENTICATION BETWEEN USERS

Multicast is widely used by enterprises, commercial stock exchanges or multimedia content delivery networks such as IPTV or military conferences. It is clear that in some of these cases an authentication between users should be required before sending any information, although this is encrypted. The aim of this section is to give another possible application of our proposal in this direction; this could be an alternative to usual certificates and offer the advantage that the server does not need to be accessed to renew a certificate during the lifetime of a user's secret key.

In fact, knowing the vector $x_1e_1 + \cdots + x_ne_n$ gives a user the opportunity to authenticate any other user trying to get information from her.

Suppose X wants to authenticate Y. Then X lets Y know $c = k^{-1}(x_1e_1 + \cdots + x_ne_n)$, where k is some random value previously generated by X. If Y is a legal user, then she would be able to get the secret k^{-1}, compute the inverse and send $c' = k(x_1e_1 + \cdots + x_ne_n)$ to X, who can now retrieve k and check that it is the right value. It is clear that these "certificates" expire with any refreshment of the session key that affects also the basis $B' = \{x_1e_1, \ldots, x_ne_n\}$. We could also make this process independent of the refreshing key process and have a second orthogonal system dedicated to the authentication between users, depending on the application or scenario being considered.

IV. EFFICIENT IMPLEMENTATION OF THE PROPOSED METHOD

This section is devoted to provide an efficient implementation of the proposed protocol. Multicore platforms are driving a new direction in software development where multithreaded applications are elected as the style-to-follow development techniques. The use of hardware accelerators (i.e. General Purpose Graphic Processing Units, GPU from now) may be considered as another strong point in the implementation of the solution. The Java programming language was selected to develop and deploy the solution because it is object oriented, which is an important issue if considering

parallel schemes as this methodology prints a natural parallel communication pattern due to inter-object message driven communication. As the multicore platform is involved, the threading part of the language is also an important key. In this case, Java is a language but is also a virtual machine, so threads are not native, this means that the threading package model that Java uses is on the user side and is a light-weight package, which is directly translated into fast context-switches and user control.

A. Implementation details

The application was built using three key objects, *the Key Sharing Framework object* (KSF from now), *a Server object* and a *Client object*. The server object is the hotspot in terms of computation due to the huge matrix that it hosts (named the vector space). The KSF object builds the framework, starts the server, manages clients and interfaces the GPU device, if present. In order to deal with such a huge matrix, EJML libraries were selected due to a better behaviour when compared with the rest of existing frameworks. From EJML we borrowed the condensed matrix navigation [1] and seized the computer's cache and the small memory foot print when managing huge and dense matrix data structures.

The implementation was divided in several stages:

- *Vector space setup*: The KSF object creates the 2D matrix. This stage is decomposed into two important steps: *orthonormalization* (that suffers from strong data dependencies) and *denormalization* used to accomodate the 2D matrix to the key sharing protocol. The *Semantic Vectors* [16] package provided us with accelerated techniques (from 10% to 20%) to quickly build the initial matrix, prior to orthogonalization.
- *Coder generation*: When the vector space is ready, then it can be reduced by column order into a 1D vector (using the addition). This 1D key is used by the Server Object to code the content to be distributed. Whenever any user is removed or a security issue is reported, this key must be refreshed. If a single user (or several users) is removed, its vector must be replaced in the vector space and this step repeated. If a security issue is reported, then the whole bunch of vectors is replaced, so the content generator coder is refreshed. This stage is clearly threadable. To avoid denial of service (DoS) attacks, the system processes refreshments by periods closing those clients that persist in a logout–login procedure.
- *User/Client login*: this stage creates data structures for each user that may claim a key to decode content provided by the server. A previous password based authentication mechanism gives credit to the
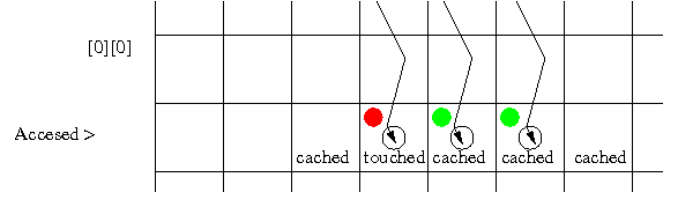

Fig. 1.   Threading the content generator coder

connection. Once the client is authorised to log in, we provide it with the key to decode messages. This stage is clearly threadable.
- *Server initialization and startup*: The KSF framework opens the server to accept requests from clients.

All the computational workload was evenly distributed between cores by the Operating System (OS in the following). If we consider that we are operating on a Java Virtual Machine environment, we can assume that the OS is doing a coarse-grain distribution of tasks between cores. And it will be good for the OS to create threads that can be identified as independent workload units that can be attached to different cores.

We used a pool of threads, to enable the threading, and a pool of tasks, to enable concurrence. A new method type, the *Task method*, was created. If the body of the method was marked as *Runnable*, the method could be interpreted as a separate unit (like an object), so it could be threaded; the *content generator coder* was threaded in this way. This method uses a vector where each component is the reduction of the corresponding column in the 2D matrix. Therefore a task was built to traverse a matrix in column order (our 2D Matrix was managed by a protected ArrayList, as adviced in [12]) and as many tasks of this type were created as the columns we configured. If threads are sharing processor's caches then this method seizes the cache when reading components, see Figure 1.

V. Results

The User login procedure was threaded using the ArrayList data structure to retrieve a row for each user, these operations are independent and threading is not penalized. Each request sent by a user was queued into the list of tasks, it is possible therefore to process tasks as soon as they come or we can choose to process them in batches; so DoS attacks can be easily avoided. Another benefit from our task list is that it is an heterogeneous task list, any method can be inserted by simply renaming it as a *T*ask method. This version still has a difficult issue to be solved, the sequential orthogonalization process; each vector (k) to be orthogonalized (using Gram-Schmidt) depends on the orthogonalization of the previous k-1 vectors. The results obtained for the threaded

TABLE I
EXECUTION OF THE PROTOCOL IN ITS THREADED VERSION (TIME IN MS)

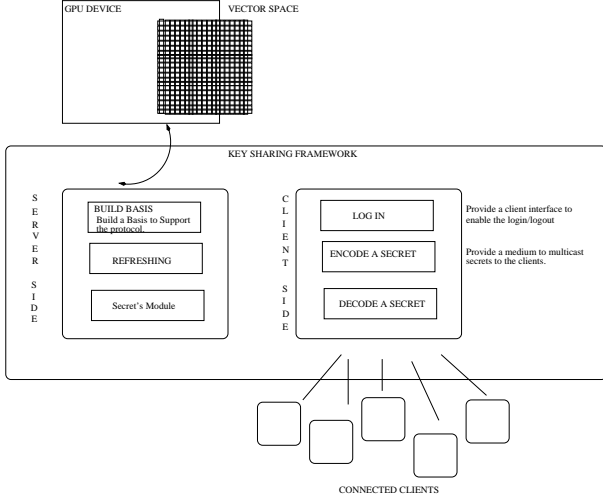| stage	core i7 extreme edition (12 hw threads, 6 cores)			dual core T9500 (2 hw threads; 2 cores)		
	5000v x 5000u	10000v x 5000u	10000v x 10000u	5000v x 5000u	10000v x 5000u	10000v x 10000u
Orthogonalization	114240	913866	913596	169909	1339753	1371083
Key Refreshment	30	197	198	66	231	235
Generator Coder	30	197	198	66	231	235
Server activation	0	1	0	2	1	3
Client's setup	1	1	2	46	80	168
Bcast	12	16	33	1	1	0
Client removal + refresh	1	1	1	45	300	556

Fig. 2.   Cuda implementation schema

version of the orthogonalization were not scaling, so, as this process is calculated only the first time the KSF framework is started, we first decided to keep it single threaded, unless a GPU device is present.

Accelerator based orthogonalization: Letting the main processing unit manage the KSF object and the Server object derive its computation from a GPU device that may act as a back-end service, might allow the system to scale. In this implementation jcuda [4] was used to interface the GPU and impersonate it as a new computational object to which we were able to send computational requests, see Figure 2. The NVIDIA GPU used was a GForce GTX-460, which is a Fermi's architecture graphics card. To build the vector space Matrix in the GPU device, we built it in the CPU and sent the data to the NVIDIA device which was a bad idea because the program was not experiencing any improvement due to the latencies, so finally a kernel was written to allocate space for the 2D matrix and populate it. Using the GPU we reduced the time for orthogonalization of approximately 90%. This section exposes the results obtained with the implementation of the proposed key sharing protocol. Table I shows the timings in milliseconds for protocol stages. An Intel Core-i7 Extreme-Edition processor was used as

our testbed. The GPU was a NVIDIA GForce GTX 460, used to accelerate the orthogonalization process up to a 90%. Table I shows the execution of the cases where we used a vector space of 10000 vectors. Tests were run using vector spaces from 10 up to 10000. Only the most representative results were shown. In this case it is significant that, as mentioned before, the real hotspot is the orthogonalization, which, as expected, has the worst times as the 2D matrix scales. The following stages were threaded with java threads, the thread pool was designed to have *12 java threads – matching with hardware threads*. The Key refreshment stage updates the vector space row by row by multiplying them by random numbers, each row is therefore converted into a potential key for a client. The Content Generator coder stage is the phase where the vector space is traversed column by column in order to calculate the reduction. Table I shows that both the key refreshment and the generation of the content coder have similar times, which means that the threading is acting properly, see the case in Table I with 10000 vectors and 5000 users and the case of 10000 vectors and 10000 users, where time is similar due to the pool of threads. To test the architectural benefits of the core i7, Table I, we launched the server in a conventional processor: core 2 duo. Although the trend reflected in Table I for the dual core case follows those studied in the core i7 case, this architecture (core 2 duo) is a laptop processor's architecture so we can see how the problem scales well even in a laptop. The timings are below the timings of the core-i7 because of the number of hardware threads per core, which is one in the core 2 duo, so no native concurrence is available.

As for the design of the pool of threads, only two active threads are running in any instant of time. Another significant difference is the bus (which is much slower than the QPI used for the processor in Table I, core i7 section). But even under those circumstances, this architecture could be used as the server for the protocol.

One of the key aspects of the implemented problem is the client's removal and the time it takes to renew its key so that it can be used by a new client logged into the system. This time is reflected in the stage named *Client's*

setup. As it can be seen, the time to refresh a client is not dependent on the size of the problem. Threads can help this operation to scale if the server finds bursts of client's removal operations.

VI. Conclusions

We have introduced a new protocol for managing keys in a centralized secure multicast setting. This protocol is shown to be secure against possible inner and/or outer attacks and it is scalable. We also showed its advantages with respect to other existing methods for key management in secure multicast schemes and we provided an efficient implementation. The method was shown to scale.

Acknowledgment

References

[1] H. Ardnt *The Universal Java Matrix Package: Everything is a Matrix!*, ICML/MLOSS, Haifa, 2010-06-25.

[2] G. Chiou and W. Chen, *Secure Broadcasting using the Secure Lock*, IEEE Transactions on Software Engineering **15**(8), 1989, 929–934.

[3] R. Di Pietro and L.V. Mancini, *Efficient and Secure Key Management for Wireless Mobile Communications*, Proceedings of the 2nd ACM international workshop on Principles of mobile computing, 2002, 66–73.

[4] W. Fan, X. Chen and X. Li, *Parallelization of RSA Algorithm Based on Compute Unified Device Architecture*, 9th International Conference on Grid and Cooperative Computing (GCC), 2010, 174–178.

[5] A. Iosevich and S. Senger, *Orthogonal Systems in Vector Spaces over Finite Fields*, The Electronic Journal of Combinatorics **15**(R151), 2008, 1–10.

[6] B. Liu, W. Zhang and T. Jiang, *A Scalable Key Distribution Scheme for Conditional Access System in Digital Pay-TV System*, IEEE Consumer Electronics **50**(2), 2004, 632–637.

[7] P.S. Kruus and J.P. Macker, *Techniques and Issues in Multicast Security*, Proceedings of Military Communications Conference (MILCOM), 1998, 1028–1032.

[8] J.A.M. Naranjo, N. Antequera, L.G. Casado and J.A. Lopez-Ramos, *A Suite of Algorithms for Key Distribution and Authentication in Centralized Secure Multicast Environments*, To appear in Journal of Computational and Applied Mathematics, doi:10.1016/j.cam.2011.02015.

[9] A. Perrig, D. Song, and J.D. Tygar, *Elk, a New Protocol for Efficient Large-Group Key Distribution*, Proceedings of IEEE Symposium on Security and Privacy (S&P), 2001, 247–262.

[10] S. Rafaeli and D. Hutchison, *A Survey of Key Management for Secure Group Communication*, ACM Computing Surveys, **35**(3), 2003, 309-329.

[11] O. Scheikl, J. Lane, R. Boyer and M. Eltoweissy, *Multi-Level Secure Multicast: the Rethinking of Secure Locks*, Proceedings of International Conference on Parallel Processing Workshop, 2002, 17–24.

[12] N. Shavit *Data Structures in the Multicore Age*, Communications of the ACM **54**(3), 2011, 76–84

[13] A.T. Sherman and D.A. McGrew, *Key Establishment in Large Dynamic Groups using One-Way Function Trees*, IEEE Transactions on Software Engineering **29**, 2003, 444–458.

[14] L.A. Vinh, *On the Number of Orthogonal Systems in Vector Spaces over Finite Fields*, The Electronic Journal of Combinatorics **15**(N32), 2008, 1–4.

[15] D. Wallner, E. Harder and R. Agee, *Key Management for Multicast: Issues and Architectures*, RFC 2627, 1999.

[16] D. Widdows and T. Cohen, *The Semantic Vectors Package: New Algorithms and Public Tools for Distributional Semantics*, IEEE 4th International Conference on Semantic Computing (ICSC), 2010

[17] C.K. Wong, M. Gouda, and S.S. Lam, *Secure Group Communications using Key Graphs*, IEEE/ACM Transactions on Networking **8**(1), 2000, 16–30.